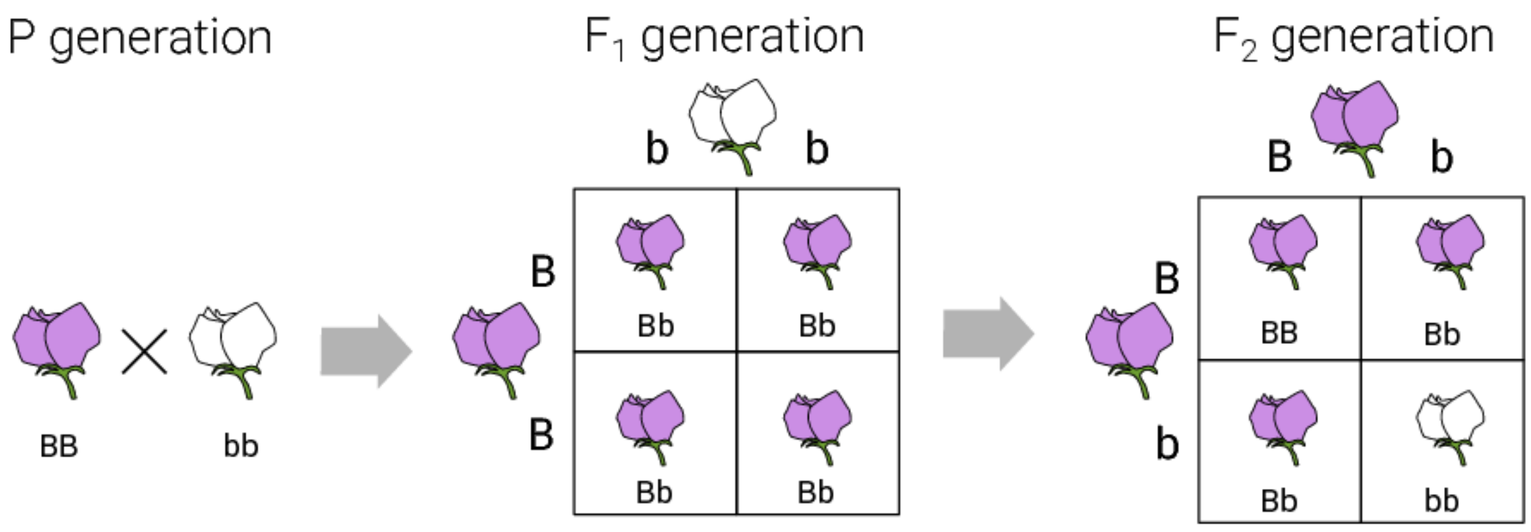
BIOS1100 H17 uke 7

Lex Nederbragt



Ukens forelesning

- noen praktiske ting
- nytt stoff denne uken
- utvalgte øvelser

Noen praktiske ting

Uke 41 (kursuke 8)

- undervisningsfri uke
- ingen forelesning
- utvidede snublegrupper
    - tirsdag 10. oktober kl 10:15-14:00 seminarrom 3508
    - torsdag 12. oktober kl 10:15-14:00 seminarrom 4619

Noen praktiske ting

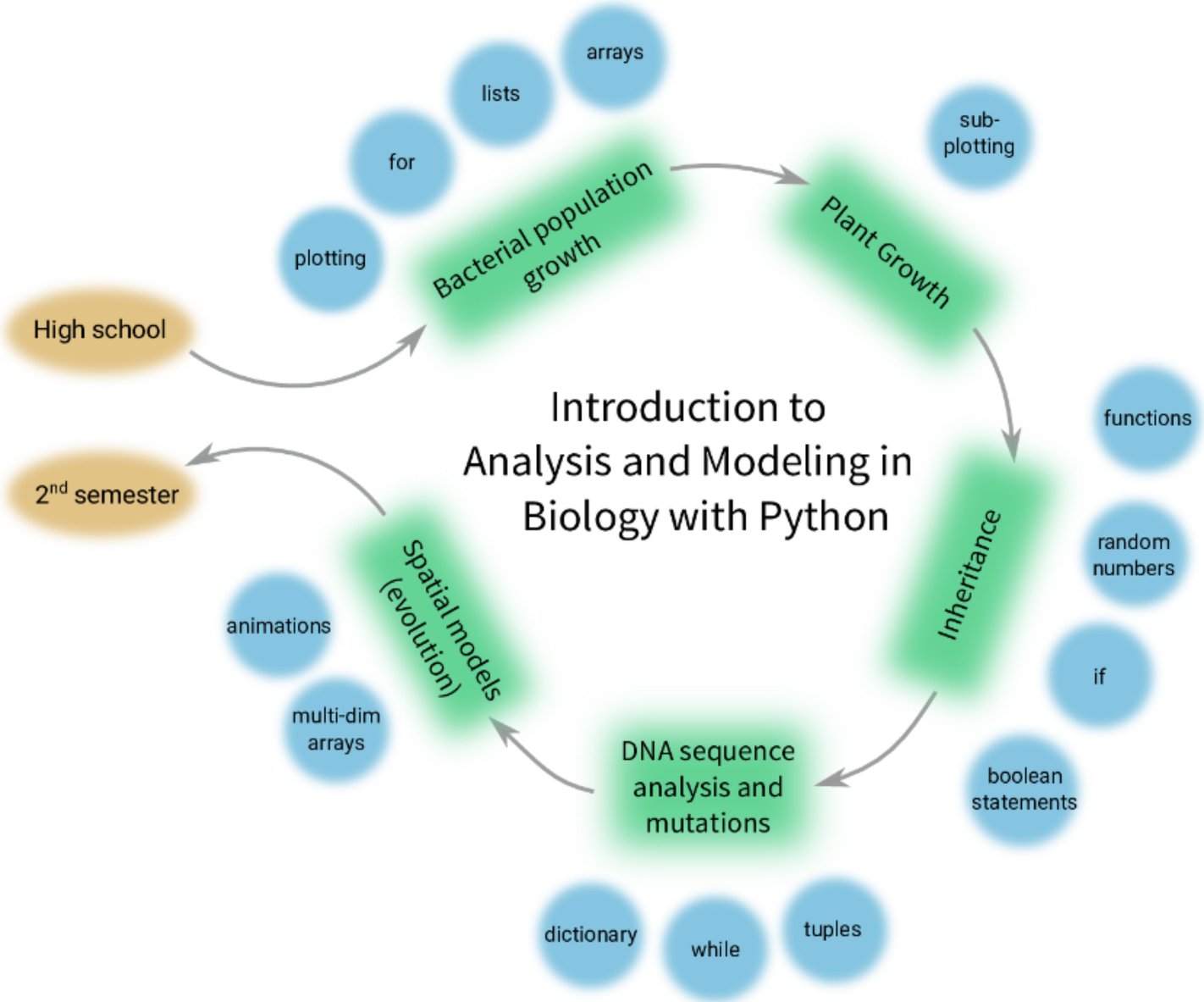
Obliger:

- tirsdag 3. oktober 23:59: frist oblig for uke **6** (kalenderuke 39)
- tirsdag 10. oktober 23:59: frist oblig for uke **7** (uke 40, denne uken)
- *ingen* oblig for uke **8** (kalenderuke 41)
- tirsdag 24. oktober 23:59: frist oblig for uke **9** (kalenderuke 42)

Noen praktiske ting

- Turtle konkuranse!
    - send din turtle tegning som notebook til bios1100@ibv.uio.no
    - den mest kreative får en pris!
    - (gruppe)lærere er juryen
    - kodekvalitet er ikke viktig

Undervisningsplan



Læringsmål denne uke

Biology

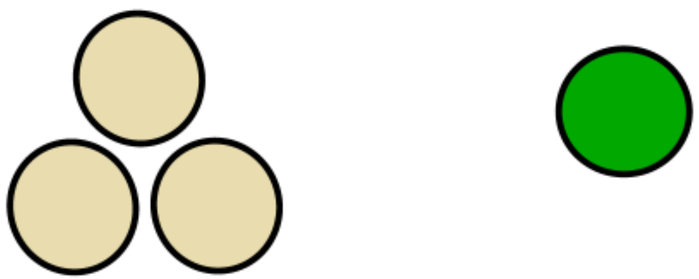
- kjenne antagelser for Mendels arvelover
- kunne lage Punnett diagrammer for hånd
- kunne sannsynlighetsteorien for tilfeldig parring
- kunne modellere tilfeldig parring

Programmering

- tilfeldig valg med `choice`
- funskjoner
- `if` tester og 'boolean statements'

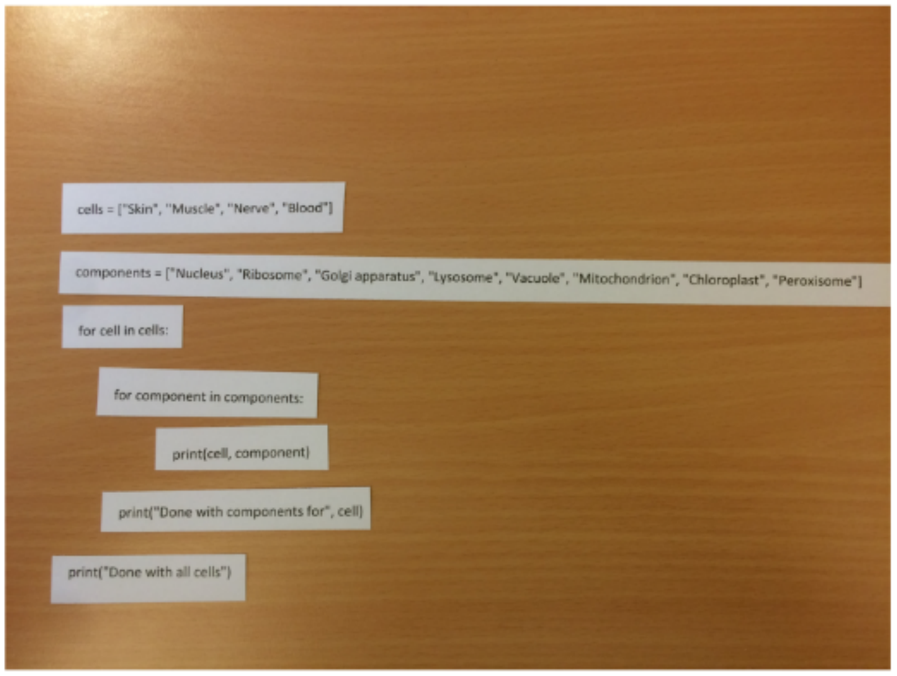
Ratio

$\left(\frac{3}{4}\right)$ versus $\left(\frac{1}{4}\right)$



3 : 1

Kombinere lister



Skin Nucleus
Skin Ribosome
Skin Golgi apparatus
Skin Lysosome
Skin Vacuole
Skin Mitochondrion
Skin Chloroplast
Skin Peroxisome
Done with components for Skin
Muscle Nucleus
Muscle Ribosome
Muscle Golgi apparatus
Muscle Lysosome
Muscle Vacuole
Muscle Mitochondrion
Muscle Chloroplast
Muscle Peroxisome
Done with components for Muscle
Nerve Nucleus
Nerve Ribosome
Nerve Golgi apparatus
Nerve Lysosome
Nerve Vacuole
Nerve Mitochondrion
Nerve Chloroplast
Nerve Peroxisome
Done with components for Nerve
Blood Nucleus
Blood Ribosome
Blood Golgi apparatus
Blood Lysosome
Blood Vacuole
Blood Mitochondrion
Blood Chloroplast
Blood Peroxisome
Done with components for Blood
Done with all cells

Kombinere lister

```
cells = ["Skin", "Muscle", "Nerve", "Blood"]
components = ["Nucleus", "Ribosome", "Golgi apparatus", "Lysosome", "Vacuole",
"Mitochondrion", "Chloroplast", "Peroxisome"]
```

Kombinere lister

```
cells = ["Skin", "Muscle", "Nerve", "Blood"]
components = ["Nucleus", "Ribosome", "Golgi apparatus", "Lysosome", "Vacuole",
"Mitochondrion", "Chloroplast", "Peroxisome"]

for cell in cells:
    for component in components:
        print(cell, component)
    print("Done with components for", cell)
print("Done with all cells")
```

Kombinere lister

```
pure_violet_flower = ["B", "B"]
pure_white_flower = ["b", "b"]

print("F1 generation:")

for allele_1 in pure_violet_flower:
    for allele_2 in pure_white_flower:
        print("Possible offspring:", allele_1, allele_2)
```

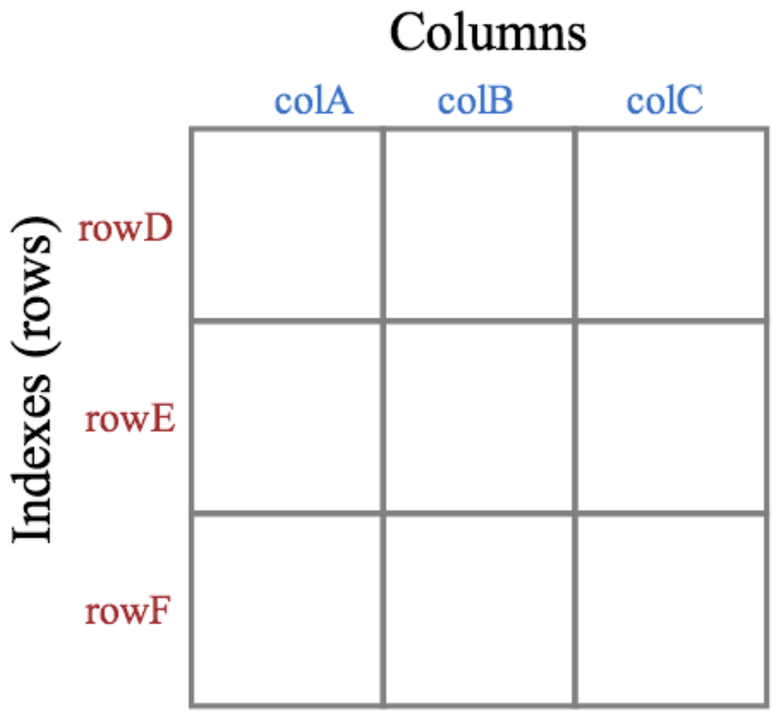
```
F1 generation:
Possible offspring: B b
Possible offspring: B b
Possible offspring: B b
Possible offspring: B b
```

Kombinere lister

```
crossbred_flower = ["B", "b"]

print("F2 generation:")

for allele_1 in crossbred_flower:
    for allele_2 in crossbred_flower:
        print("Possible offspring:", allele_1, allele_2)
```

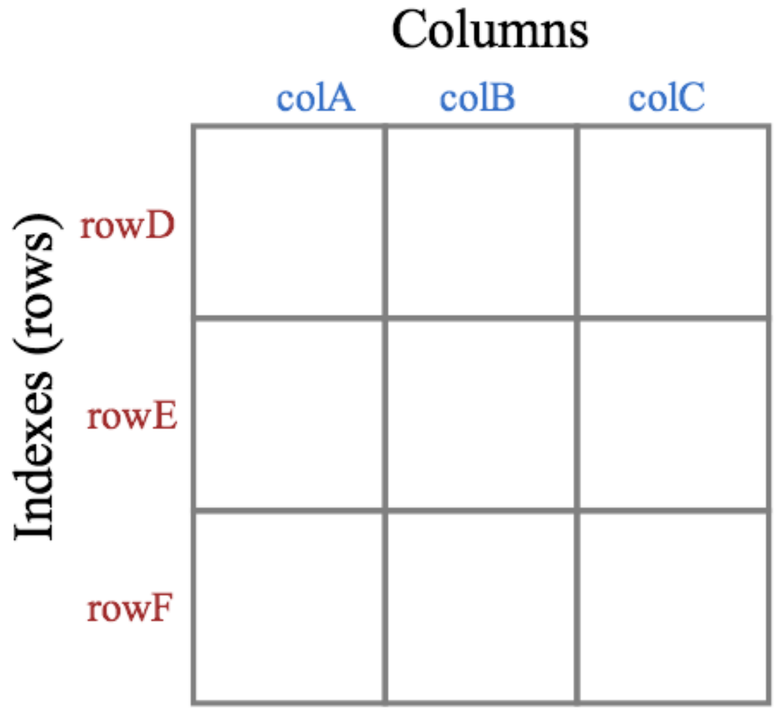
```
F2 generation:
Possible offspring: B B
Possible offspring: B b
Possible offspring: b B
Possible offspring: b b
```

Pandas dataframe



Pandas dataframe

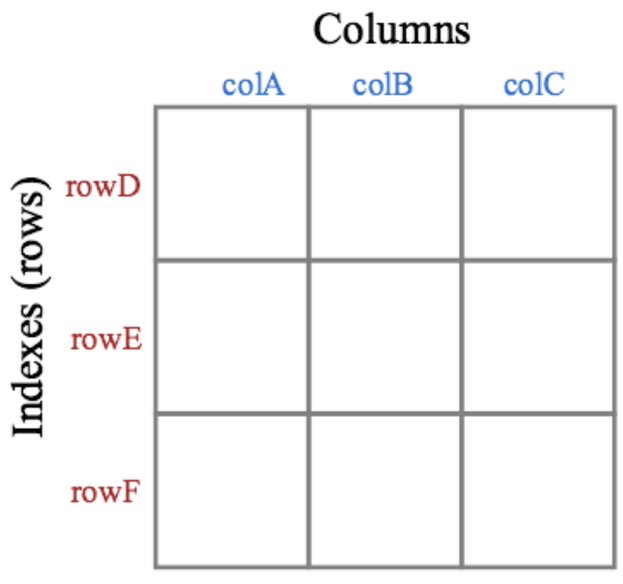
```
col_names = ["colA", "colB", "colC"]
row_names = ["rowD","rowE","rowF"]
df_example = pandas.DataFrame(index=row_names, columns=col_names)
```



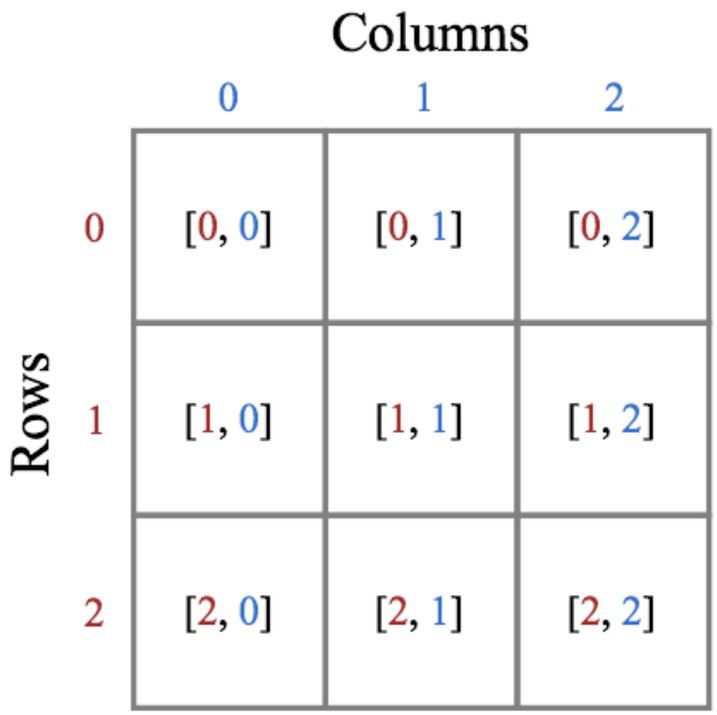
Pandas dataframe

```
col_names = ["colA", "colB", "colC"]
row_names = ["rowD","rowE","rowF"]
df_example = pandas.DataFrame(index=row_names, columns=col_names)
print(df_example)
```

```
      colA colB colC
rowD   NaN  NaN  NaN
rowE   NaN  NaN  NaN
rowF   NaN  NaN  NaN
```



Pandas dataframe



Row-by-column indexing

Pandas dataframe

Fylle tabellen:

```
iloc[row, column]
```

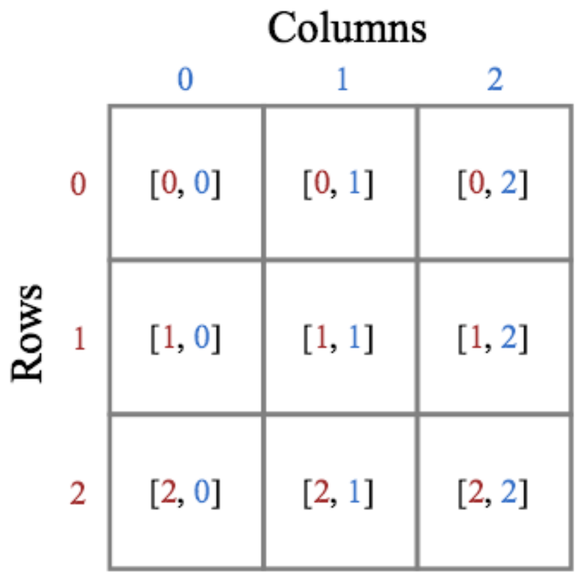
Rows \ Columns	0	1	2
0	[0, 0]	[0, 1]	[0, 2]
1	[1, 0]	[1, 1]	[1, 2]
2	[2, 0]	[2, 1]	[2, 2]

Pandas dataframe

For eksempel:

```
df_example.iloc[0, 0] = "firstcell"
print(df_example)
```

```
       colA        colB colC
rowD  firstcell   NaN   NaN
rowE         NaN   NaN   NaN
rowF         NaN   NaN   NaN
```



Python `enumerate` funksjonen

En vanlig for løkke:

```
my_list = ["A", "B", "C", "D"]
for element in my_list:
    print("element er", element)
```

```
element er A
element er B
element er C
element er D
```

Python `enumerate` funksjonen

Men hvordan å få denne outputen?

```
index 0 er element A
index 1 er element B
index 2 er element C
index 3 er element D
```

Python `enumerate` funksjonen

Men hvordan å få denne outputen?

```
index 0 er element A
index 1 er element B
index 2 er element C
index 3 er element D
```

Mulighet 1:

```
my_list = ["A", "B", "C", "D"]

for index in range(4):
    print("index", index, "er element", my_list[index])
```

Python `enumerate` funksjonen

Men hvordan å få denne outputen?

```
index 0 er element A
index 1 er element B
index 2 er element C
index 3 er element D
```

Mulighet 2, mer generelt:

```
my_list = ["A", "B", "C", "D"]

for index in range(len(my_list)):
    print("index", index, "er element", my_list[index])
```

Python `enumerate` funksjonen

Men hvordan å få denne outputen?

```
index 0 er element A
index 1 er element B
index 2 er element C
index 3 er element D
```

Mulighet 3, med `enumerate`

```
my_list = ["A", "B", "C", "D"]

for index, element in enumerate(my_list):
    print("index", index, "er element", element)
```

Python enumerate funksjonen

```
my_list = ["A", "B", "C", "D"]

for index, element in enumerate(my_list):
    print("index:", index, ", element:", element)
```

```
index 0 er element A
index 1 er element B
index 2 er element C
index 3 er element D
```

Pandas dataframe

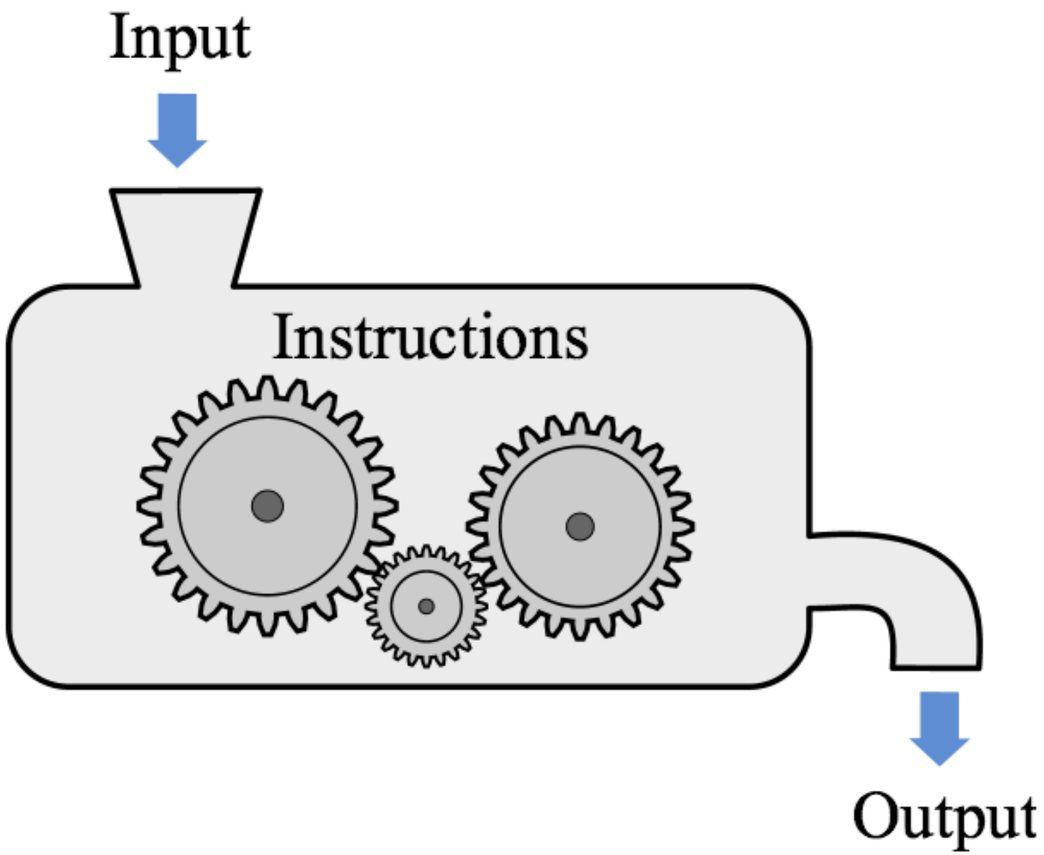
```
col_names = ["colA", "colB", "colC"]
row_names = ["rowD","rowE","rowF"]

for col, col_name in enumerate(col_names):
    for row, row_name in enumerate(row_names):
        print(col, col_name, row, row_name, col_name+row_name)
```

```
0 colA 0 rowD colArowD
0 colA 1 rowE colArowE
0 colA 2 rowF colArowF
1 colB 0 rowD colBrowD
1 colB 1 rowE colBrowE
1 colB 2 rowF colBrowF
2 colC 0 rowD colCrowD
2 colC 1 rowE colCrowE
2 colC 2 rowF colCrowF
```

Pandas dataframe

```
col_names = ["colA", "colB", "colC"]
row_names = ["rowD","rowE","rowF"]

for col, col_name in enumerate(col_names):
    for row, row_name in enumerate(row_names):
        df_example.iloc[row, col] = col_name+row_name
print(df_example)
```

```
      colA      colB      colC
rowD  colArowD  colBrowD  colCrowD
rowE  colArowE  colBrowE  colCrowE
rowF  colArowF  colBrowF  colCrowF
```

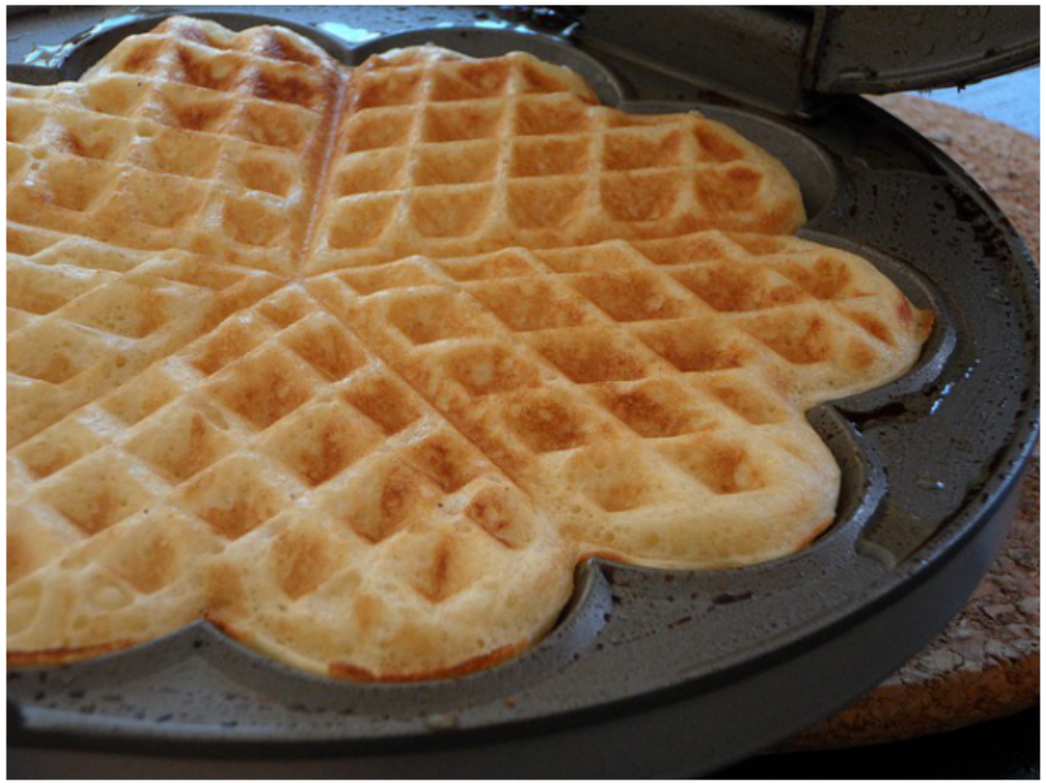

Input
Instructions
Output

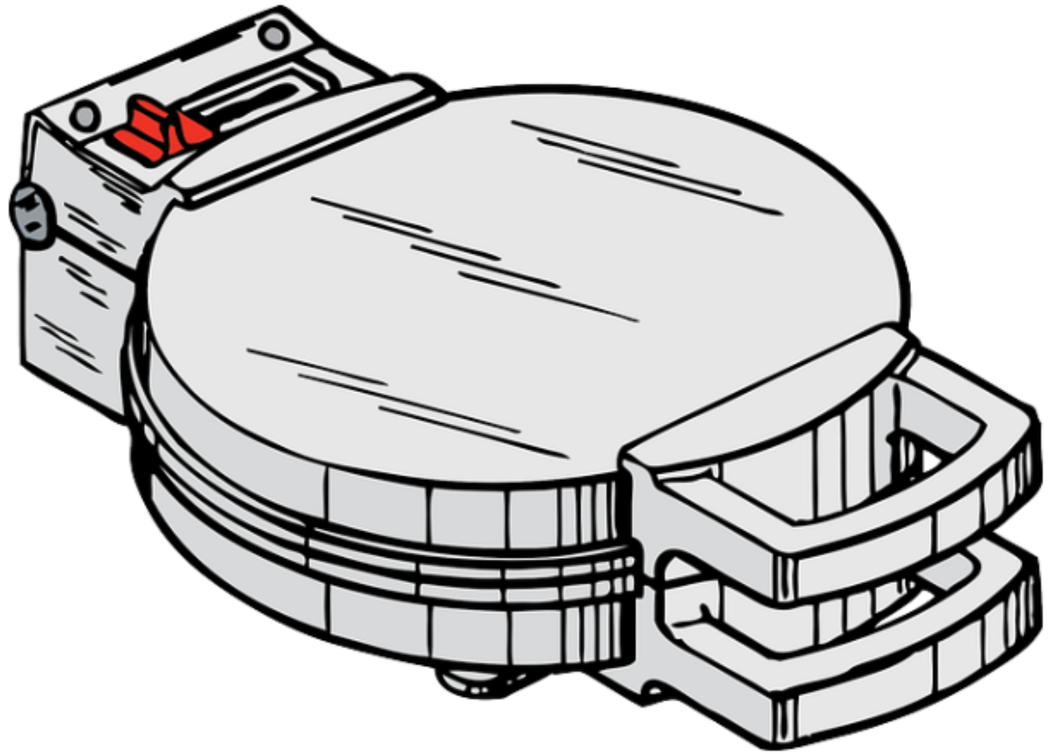
Repetisjon

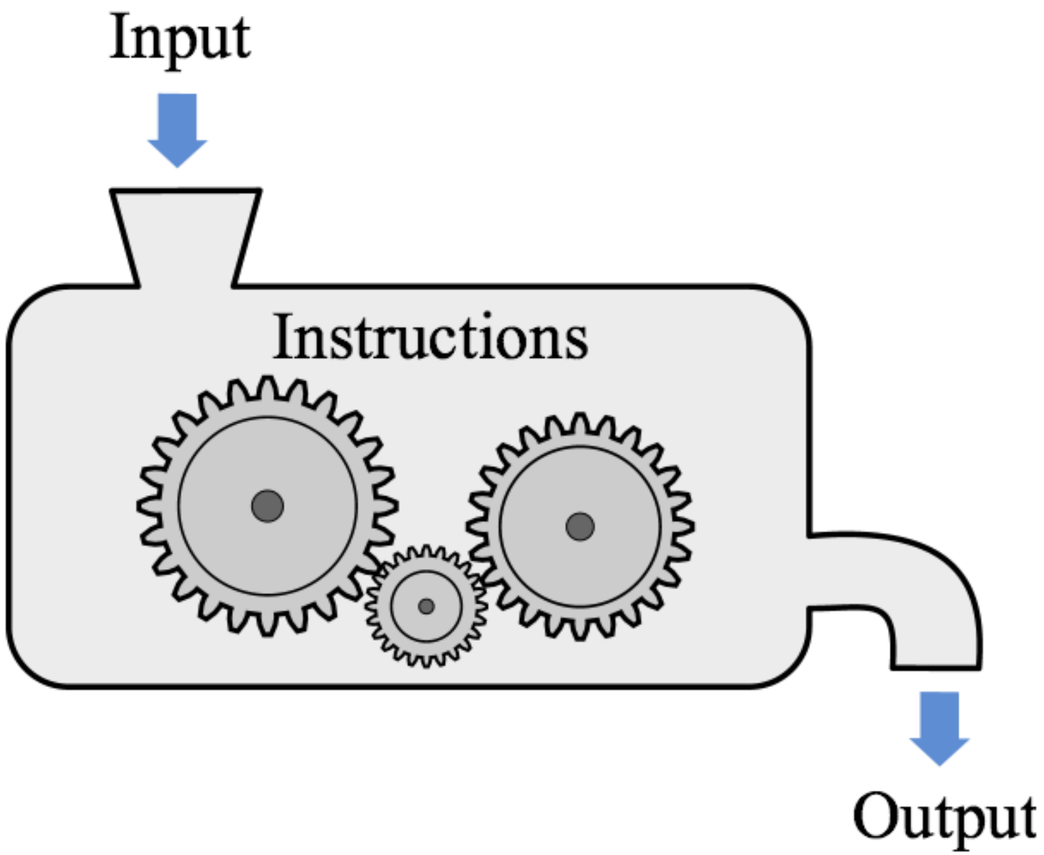
Rule of three.
In programming, there is a rule of thumb called ''the rule of three''. It states that if a task has to be repeated three times or more, we should automate it.

Funksjoner

Hva er funksjonen til et waffeljern?






Input
Instructions
Output

Funksjoner

```
celsius = (fahrenheit - 32)/1.8
```

Funksjoner

```
celsius = (fahrenheit - 32)/1.8
```

```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32)/1.8
    return celsius
```

Funksjoner

```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32)/1.8
    return celsius
```

```
print(fahrenheit_to_celsius(10))
```

```
-12.222222222222221
```

Funksjoner

```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32)/1.8
    return celsius

T_fahrenheit = 10
T_celsius = fahrenheit_to_celsius(T_fahrenheit)
print(T_fahrenheit, "degrees Fahrenheit is", T_celsius, " degrees celsius.")
```

```
10 degrees Fahrenheit is -12.222222222222221  degrees celsius.
```

Funksjoner

```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32)/1.8
    return celsius

bananas = 10
volkswagen = fahrenheit_to_celsius(bananas)
print(bananas, "degrees Fahrenheit is", volkswagen, " degrees celsius.")
```

Funksjoner

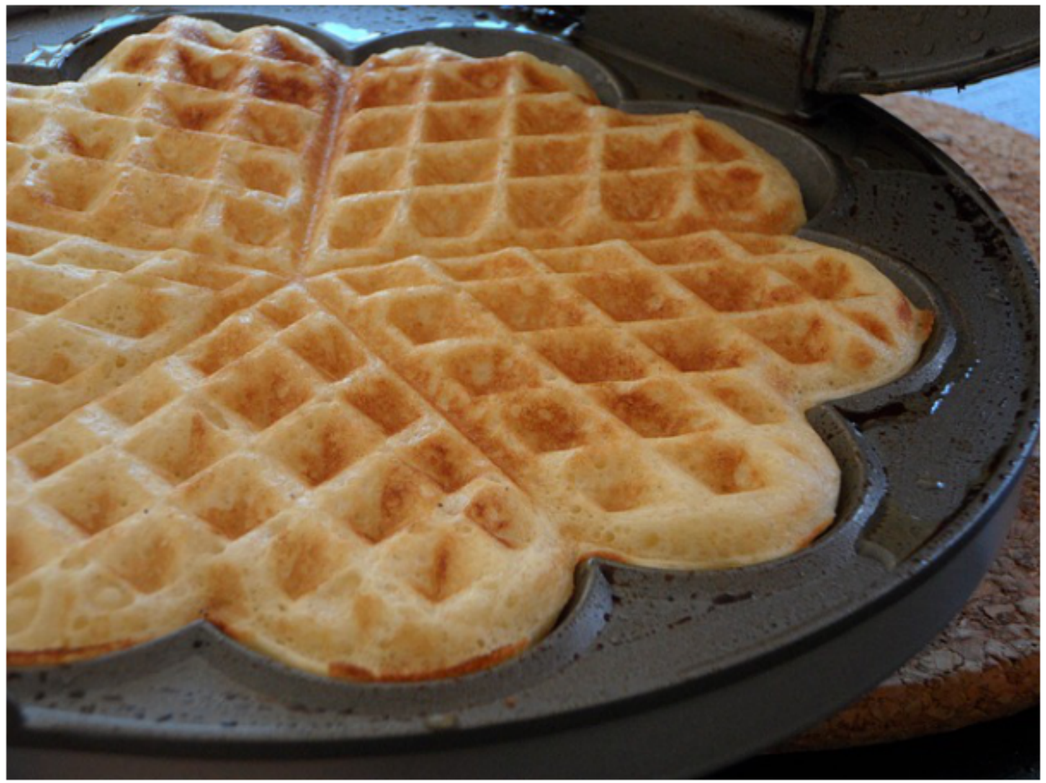
```
def hello(darwin):
    dog = (darwin - 32)/1.8
    return dog

bananas = 10
volkswagen = hello(bananas)
print(bananas, "degrees Fahrenheit is", volkswagen, " degrees celsius.")
```

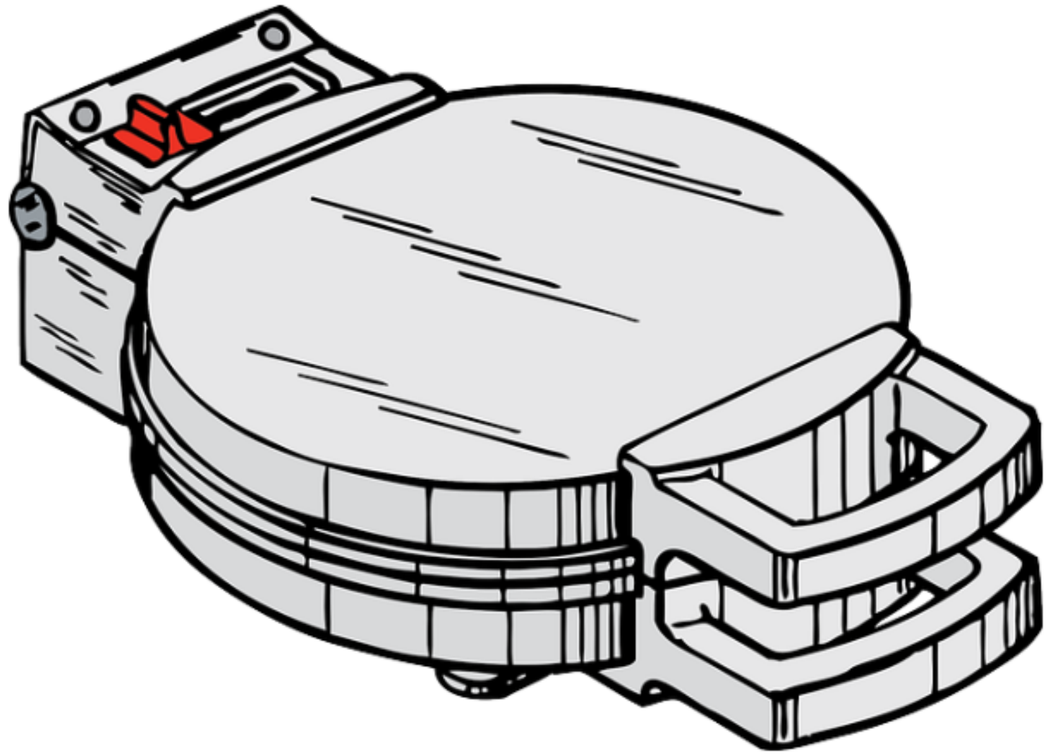
Funksjoner

```
def fahrenheit_to_celsius(fahrenheit):
    celsius = (fahrenheit - 32)/1.8
    return celsius

fahrenheit = 10
celsius = fahrenheit_to_celsius(fahrenheit)
print(fahrenheit, "degrees Fahrenheit is", celsius, " degrees celsius.")
```

Funksjoner

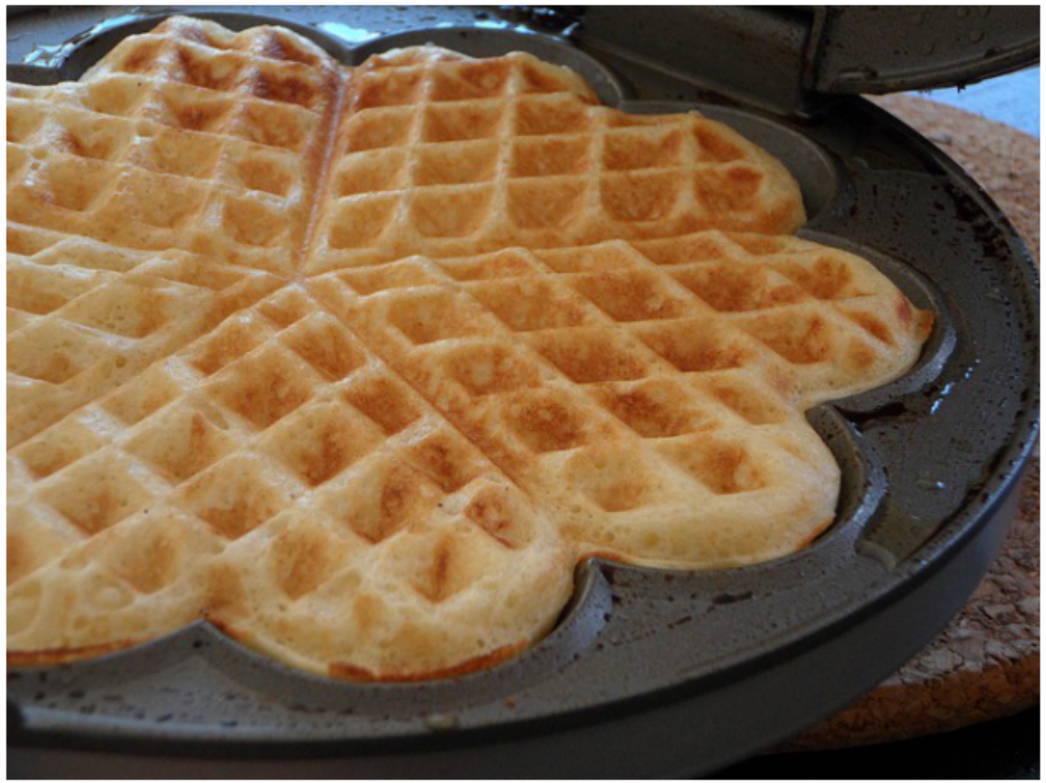
Hva er funksjonen til et waffeljern?

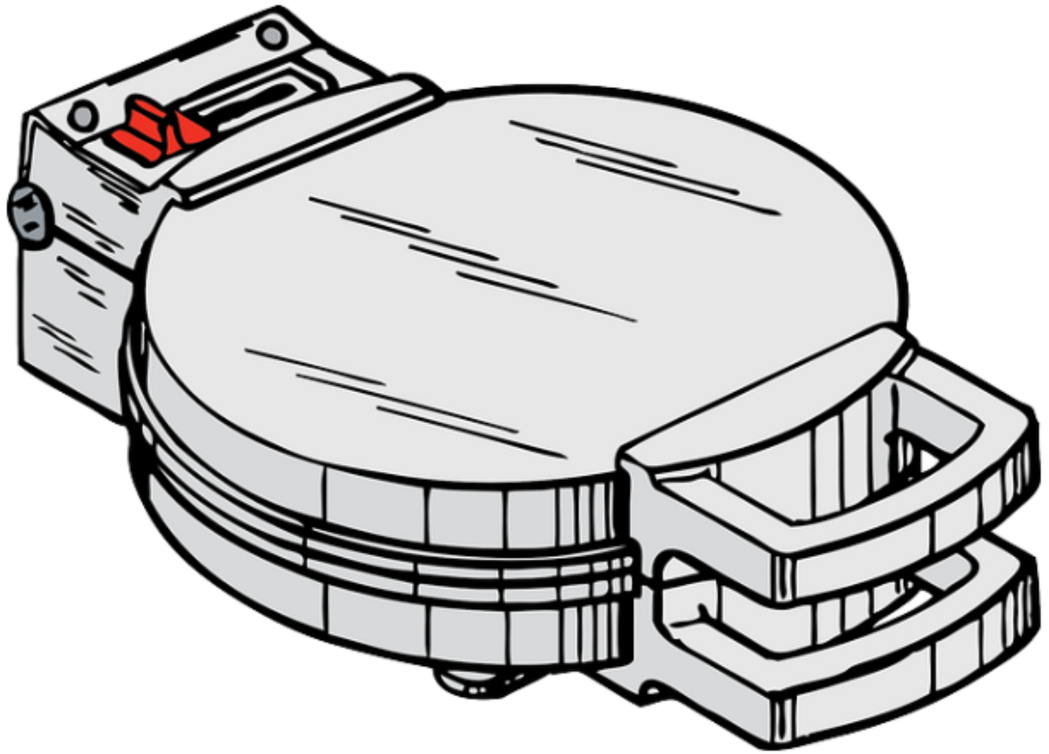




Funksjoner

Hva er funksjonen til et waffeljern?





Waffeljernet bryr seg ikke om deien!

Punnett square funksjonen

```
col_names = ["colA", "colB", "colC"]
row_names = ["rowD","rowE","rowF"]

for col, col_name in enumerate(col_names):
    for row, row_name in enumerate(row_names):
        df_example.iloc[row, col] = col_name+row_name
print(df_example)
```

```
def create_punnett_square(parent_1, parent_2):
    punnett_square = pandas.DataFrame(index=parent_1, columns=parent_2)

    for index_1, allele_1 in enumerate(parent_1):
        for index_2, allele_2 in enumerate(parent_2):
            punnett_square.iloc[index_1, index_2] = allele_1 + allele_2

    return punnett_square
```

Punnett square funksjonen

```
parent_1 = ["IG", "ig"]
parent_2 = ["IG", "ig"]

punnett_square = create_punnett_square(parent_1, parent_2)
print(punnett_square)
```

```
    IG    ig
IG  IGIG  IGig
ig  igIG  igig
```

Punnett square funksjonen

```
parent_1 = ["IG", "Ig", "iG", "ig"]
parent_2 = ["IG", "Ig", "iG", "ig"]

punnett_square = create_punnett_square(parent_1, parent_2)
print(punnett_square)
```

```
    IG    Ig    iG    ig
IG  IGIG  IGIg  IGiG  IGig
Ig  IgIG  IgIg  IgiG  Igig
iG  iGIG  iGIg  iGiG  iGig
ig  igIG  igIg  igiG  igig
```

Mendel's rules

Mendel's rules for inheritance

- An inheritable trait is carried by a gene, and the genes exist in two alleles.
- Each organism inherits two alleles, one from each parent.
- Inherited alleles are chosen at random.
- If the inherited alleles are different, one is dominant and overrules the other, which is recessive.

Mendel's rules

Mendel's rules for inheritance

- An inheritable trait is carried by a gene, and the genes exist in two alleles.

```
parent_1 = ["B", "b"]
```

Mendel's rules

Mendel's rules for inheritance

- Each organism inherits two alleles, one from each parent.

```
parent_1 = ["B", "b"]
parent_2 = ["B", "b"]
```

Mendel's rules

Mendel's rules for inheritance

- Inherited alleles are chosen at random.

Tilfeldig valg

```
from pylab import *

colors = ["red", "green", "blue", "violet"]
color = choice(colors)

print(color)
```

Mendel modellen

```
parent_1 = ["B", "b"]
parent_2 = ["B", "b"]

allele_1 = choice(parent_1)
allele_2 = choice(parent_2)

genotype = [allele_1, allele_2]

print("The genotype is:", genotype)
```

Mendel's rules

Mendel's rules for inheritance

- Inherited alleles are chosen at random.

```
allele_1 = choice(parent_1)
allele_2 = choice(parent_2)
```

Mendel's rules

Mendel's rules for inheritance

- If the inherited alleles are different, one is dominant and overrules the other, which is recessive.

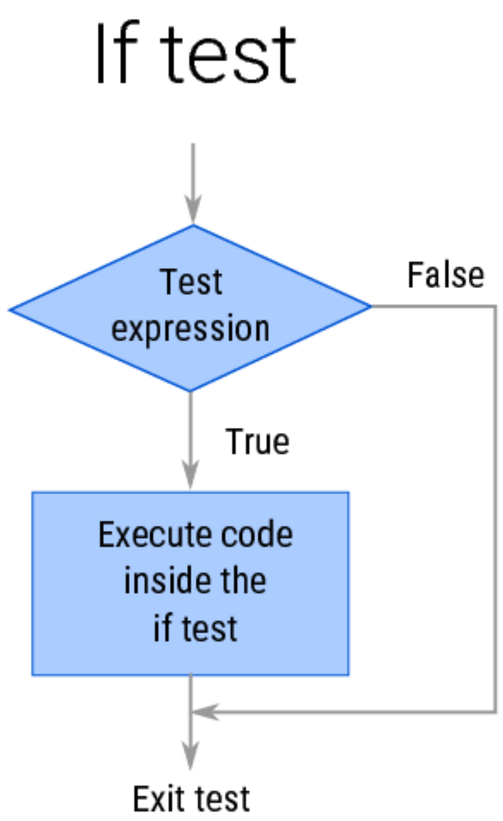
Mendel modellen

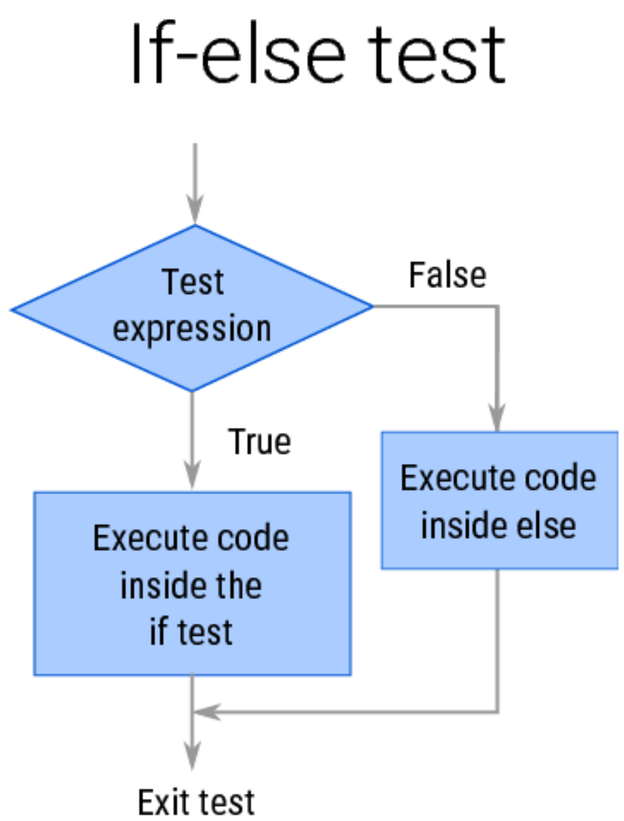
```
parent_1 = ["B", "b"]
parent_2 = ["B", "b"]

allele_1 = choice(parent_1)
allele_2 = choice(parent_2)

genotype = [allele_1, allele_2]

print("The genotype is:", genotype)
```

Mendel modellen

```
parent_1 = ["B", "b"]
parent_2 = ["B", "b"]

allele_1 = choice(parent_1)
allele_2 = choice(parent_2)

genotype = [allele_1, allele_2]

print("The genotype is:", genotype)

if genotype == ["B", "B"]:
    phenotype = "violet"
if genotype == ["B", "b"]:
    phenotype = "violet"
if genotype == ["b", "B"]:
    phenotype = "violet"
if genotype == ["b", "b"]:
    phenotype = "white"

print("The phenotype is", phenotype)
```

if .. else


If test
Test expression
False
True
Execute code inside the if test
Exit test


If-else test
Test expression
False
True
Execute code inside the if test
Execute code inside else
Exit test

Mendel modellen

```
parent_1 = ["B", "b"]
parent_2 = ["B", "b"]

allele_1 = choice(parent_1)
allele_2 = choice(parent_2)

genotype = [allele_1, allele_2]

print("The genotype is:", genotype)

if genotype == ["b", "b"]:
  phenotype = "white"
else:
  phenotype = "violet"

print("The phenotype is", phenotype)
```

Gjøre valg - `if`

```
attendees = ['Ola','Kari','Jane','John']

if 'Kari' in attendees:
    print("Kari is coming!")
```

Mendel modellen

```
genotype = ["b", "B"]

if "B" in genotype:
    phenotype = "violet"
else:
    phenotype = "white"

print("The phenotype is", phenotype)
```

```
The phenotype is violet
```

Mendel modellen

```
genotype = ["b", "b"]

if "B" in genotype:
    phenotype = "violet"
else:
    phenotype = "white"

print("The phenotype is", phenotype)
```

```
The phenotype is white
```

Mendel's rules

Mendel's rules for inheritance

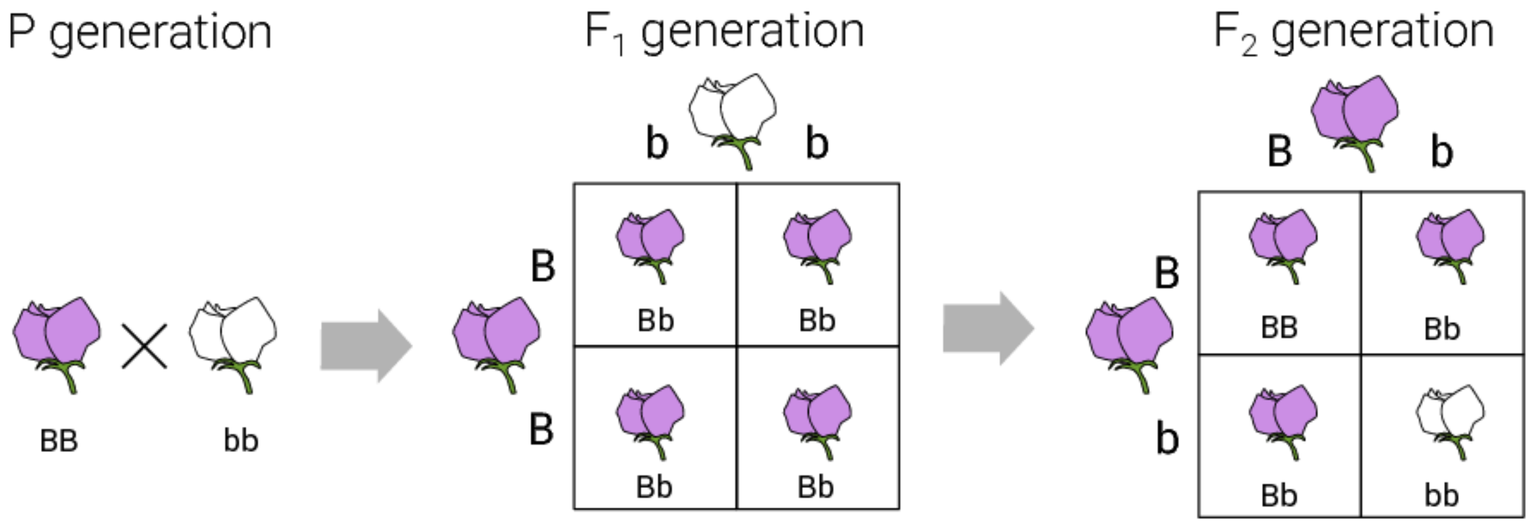
- If the inherited alleles are different, one is dominant and overrules the other, which is recessive.

```
if "B" in genotype:
    phenotype = "violet"
else:
    phenotype = "white"
```

Mendel modellen

```
parent_1 = ["B", "b"]
parent_2 = ["B", "b"]

allele_1 = choice(parent_1)
allele_2 = choice(parent_2)

genotype = [allele_1, allele_2]

print("The genotype is:", genotype)

if "B" in genotype:
    phenotype = "violet"
else:
    phenotype = "white"

print("The phenotype is", phenotype)
```

Mendel modellen



Mendel modellen

Funksjon `create_offspring`

```
def create_offspring(parent_1, parent_2):
    allele_1 = choice(parent_1)
    allele_2 = choice(parent_2)

    genotype = [allele_1, allele_2]

    return genotype
```

Mendel modellen

```
violet_flowers_count = 0
white_flowers_count = 0

P_generation_violet = ['B', 'B']
P_generation_white = ['b', 'b']

# Cross-pollinate
F1_generation = create_offspring(P_generation_violet, P_generation_white)

# Self-pollinate
F2_generation = create_offspring(F1_generation, F1_generation)

# Find the phenotype and increase correct count
if 'B' in F2_generation:
    violet_flowers_count = violet_flowers_count + 1
else:
    white_flowers_count = white_flowers_count + 1
```

Mendel modellen

```
white_flowers_count = 0
violet_flowers_count = 0

for i in range(28000):
    # True bred flowers
    P_generation_violet = ['B', 'B']
    P_generation_white = ['b', 'b']

    # Cross-pollinate
    F1_generation = create_offspring(P_generation_violet, P_generation_white)

    # Self-pollinate
    F2_generation = create_offspring(F1_generation, F1_generation)

    # Find the phenotype and increase correct count
    if 'B' in F2_generation:
        violet_flowers_count = violet_flowers_count + 1
    else:
        white_flowers_count = white_flowers_count + 1
```

Mendel modellen

```
ratio = violet_flowers_count/white_flowers_count

print("The result for the F2 was", violet_flowers_count,
      "violet flowers and", white_flowers_count, "white flowers.")
print("The ratio is", ratio, "to 1")
```

Mendel modellen

```
ratio = violet_flowers_count/white_flowers_count

print("The result for the F2 was", violet_flowers_count,
      "violet flowers and", white_flowers_count, "white flowers.")
print("The ratio is", ratio, "to 1")
```

```
The result for the F2 was 21014 violet flowers and 6986 white flowers.
The ratio is 3.008016032064128 to 1
```

Utvalgte øvelser

Bruke Google for å finne svar på spørsmålet:

> why doesn't `range` in python return a list?

Utvalgte øvelser

- Exercise 4: The Rice And Chessboard Story revisited
- Exercise 5: Rabbit population growth